Error Backpropagation

Due at 4:00pm on Wednesday 7 February 2018

What you need to get

- `YOU_a2.ipynb`: a Python notebook (hereafter called "the notebook")
- `Network.pyc`: Module with solution for `Network` class

What you need to know

This is the complete assignment, although it might be updated from time to time with clarifications – see the version number and release date at the bottom.

What to do

1. [2 marks] The logistic function is defined as

$$\sigma(z) = \frac{1}{1+e^{-z}} \ .$$

Prove that

$$\frac{d\sigma(z)}{dz} = \sigma(z)\left(1-\sigma(z)\right) \ .$$

2. [3 marks] Consider the classification problem in which you have K classes, labelled $\{1, 2, \ldots, K\}$. Suppose we have a labelled dataset,

$$\left\{ \left(\vec{x}^{(1)}, \vec{t}^{(1)}\right), \ldots, \left(\vec{x}^{(P)}, \vec{t}^{(P)}\right) \right\}_i$$

where $\vec{x}^{(i)} \in \mathbb{R}^X$ and $\vec{t}^{(i)} \in \mathbb{R}^Y$ for $i = 1, \ldots, P$. That is, $\vec{t}^{(i)}$ is the one-hot classification vector

$$t_k^{(i)} = \begin{cases} 1 & \text{if } \vec{x}^{(i)} \in C_k \\ 0 & \text{if } \vec{x}^{(i)} \notin C_k \ . \end{cases}$$

where C_k is the set of inputs that belong to class k.

Suppose you have a neural network whose operation is represented by the function f, such that for a given input $\vec{x}^{(i)}$, the output is

$$\vec{y}^{(i)} = f\left(\vec{x}^{(i)} \,|\, \theta\right)$$

where θ represents all the connection weights and biases in the network. Let us also assume that the output of the network, $\vec{y}^{(i)}$, is a probability vector corresponding to the Bernoulli distribution,

$$P\left(\vec{x}^{(i)} \in C_k \,|\, \theta\right) = \prod_j \left(\vec{y}_j^{(i)}\right)^{\vec{t}_j^{(i)}} \left(1 - \vec{y}_j^{(i)}\right)^{(1-\vec{t}_j^{(i)})}$$

Prove that the negative log-likelihood of observing sample $\left(\vec{x}^{(i)}, \vec{t}^{(i)}\right)$ is

$$-\sum_{k=1}^{K} \vec{t}_k^{(i)} \ln \vec{y}_k^{(i)} + \left(1 - \vec{t}_k^{(i)}\right) \ln\left(1 - \vec{y}_k^{(i)}\right) \ .$$

3. [4 marks] Consider the network $\vec{y} = f(\vec{x}, \theta)$ with output loss function $E\left(\vec{y}, \vec{t}\right)$, where $\vec{y}, \vec{t} \in \mathbb{R}^Y$. Let the input current to the output layer be $\vec{z}$, so that $\vec{y} = \sigma(\vec{z})$, and $\sigma(\cdot)$ is the activation function for the nodes in the output layer.

   Derive an expression for $\frac{\partial E}{\partial \vec{z}}$ for the following two combinations:

   (a) Cross entropy, and logistic activation function.

   $$E(\vec{y}, \vec{t}) = -\sum_{k=1}^{Y} t_k \ln y_k + (1 - t_k)\ln(1 - y_k)$$

   (b) Mean squared error, and linear activation function.

   $$E(\vec{y}, \vec{t}) = \sum_{k=1}^{Y} (y_k - t_k)^2$$

4. **Implementing Backpropagation**

   For this question, you must complete the implementation of the `Network` class. To do so, follow these directions:

   (a) `FeedForward`: Complete the function `FeedForward` according to the specifications in its documentation. [3 marks]

   (b) `CrossEntropy`: Complete the `CrossEntropy` function, which evaluates the cross entropy between the supplied target, and the activity of the top layer of the network. [1 mark]

   (c) `MSE`: Complete the `MSE` function, which evaluates the mean squared error between the supplied target, and the activity of the top layer of the network. [1 mark]

   (d) `Backprop`: Complete the `Backprop` function, which uses the network state (ie. after a feed-forward pass) and the corresponding target to compute the error gradients, and perform an update to the network weigths and biases. [6 marks]

   (e) `Learn`: Complete the `Learn` function, which performs gradient descent over the training dataset to try to optimize the network weights and biases. The function should perform the specified number of epochs, each time randomizing the order of the training samples (you can use `numpy.random.shuffle` for that). [4 marks]

For your convenience, the notebook includes two sample datasets for you to test your implementation on. One is a classification problem, and one is a regression problem. Your implementation should work with these lines of code. But I encourage you to tinker with them.

There is also a pre-compiled module with my implementation of the `Network` class. This might be helpful for testing, or if you have trouble with one of the earlier functions, and would like to move on and implement one of the dependent functions.

Enjoy!

What to submit

Your assignment submission should be a single jupyter notebook file, named (`<WatIAM>_a1.ipynb`), where `<WatIAM>` is your UW WatIAM login ID (not your student number). The notebook must include solutions to **all** the questions. Submit this file to Desire2Learn. You do not need to submit any of the modules supplied for the assignment.